Le gradient de THYC3D par Odyssée

Christèle Faure

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Le Gradient de THYC3D *par* Odyssée

Christèle FAURE

N° 3519

Octobre 1998

THÈME 2

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

Le Gradient de THYC3D par Odyssée

Christèle FAURE*

Thème 2 — Génie logiciel
et calcul symbolique

Projet SAFIR

Rapport de recherche n° 3519 — Octobre 1998 — 38 pages

Résumé : Odyssée est un outils de différentiation automatique de code Fortran qui met en oeuvre à la fois le mode inverse et le mode direct de différentiation automatique.

Ce document présente les résultats obtenus lors de la génération automatique d'un code inverse (adjoint) pour le code de thermohydraulique opérationnel THYC par Odyssée. Il faut noter qu'à notre connaissance, il n'existe aucune expérience de différentiation automatique de code opérationnel en mode inverse avec un autre outils automatique. En effet, les trois autres outils de différentiation automatique qui mettent en oeuvre le mode inverse (adolc, tamc, Padre2) ne peuvent pas être appliqués sur de très grands codes.

Mots-clés : Odyssee, différentiation automatique, Fortran, transformation de programme, THYC.

Ce travail a été en partie financé par Électricité de France (Contrat EDF T13/L11/IK8568/RNE651).

* Email : Christele.Faure@sophia.inria.fr, URL : http://www.inria.fr/safir/WHOSWHO/Christele.Faure

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 — Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

The gradient of THYC3D using Odyssée

Abstract: Odyssée is an automatic differentiation processor for Fortran code that implements both the forward and reverse mode of automatic differentiation.

This document presents the application of Odyssée on an operative code called THYC. On must notice that as far as we know there is no other experiment.

Key-words: Odyssee, automatic differentiation, computational differentiation, Fortran, program transformation, THYC.

Table des matières

1 Introduction **5**

2 THYC3D pour la Différentiation Automatique **7**

2.1 Utilisation implicite des pointeurs 7

2.2 Aliasing par passage de paramètres 8

2.3 Les fonctions tabulées 10

3 Dérivation manuelle des fonctions tabulées **13**

3.1 Principes généraux 13

3.2 Dérivées des fonctions simples 16

3.3 Dérivées des fonctions memo de positionnement 17

3.4 Dérivées des fonctions memo de calcul 17

4 Dérivation de THYC3D par Odyssée **21**

4.1 Dérivation de princp par rapport à parcor 23

4.2 Modification manuelle du code dérivé 24

4.3 Écriture de la routine principale 26

5 Validation des résultats **29**

5.1 Résultats des tests en valeur 31

5.2 Résultats des tests en temps et espace 33

5.3 Conclusion 35

Chapitre 1

Introduction

La différentiation automatique est une technique permettant d'obtenir les dérivées exactes (aux erreurs d'arrondi près) d'une fonction représentée par un programme informatique. Il existe deux classes d'outils de différentiation automatique [1] : ceux qui fonctionnent par surcharge d'opérateur et ceux qui fonctionnent par transformation de programme. Dans le premier cas, pour obtenir les dérivées, il faut lier le code initial dans lequel ont été ajouté des déclarations spécifiques à une bibliothèque prédéfinie (le système `adolc` fonctionne ainsi). Dans le second cas, le système génère à partir du code de la fonction un nouveau code qui calcule les dérivées et c'est ce dernier code qui doit être exécuté. L'intérêt de la seconde technique (transformation de code) est que le code qui calcule la dérivée peut être utilisé dans un autre code comme un optimiseur par exemple.

`Odyssée` [FP97] est un logiciel de différentiation automatique mettant en œuvre à la fois la dérivation en mode direct et en mode inverse.
Il est développé à l'INRIA (Sophia Antipolis) dans le projet SAFIR en Caml sous une forme modulaire. Le système `Odyssée` prend pour entrée un code écrit en Fortran-77 qui calcule une fonction différentiable par morceaux et fournit en sortie un nouveau code Fortran-77 qui calcule une dérivée tangente (une dérivée directionnelle) ou cotangente (un gradient) de cette fonction. Le système fonctionne par transformation du code source. Au cours de cette étude, le système a été modifié : certaines

1. On peut trouver dans `http://www.mcs.anl.gov/Projects/autodiff/AD_Tools/ad_tools.html` une description de tous les systèmes de différentiation automatique existants.

limitations sur le code source ont été levées et des algorithmes nouveaux de différentiation automatique ont été spécifiés et implémentés. Ceci est décrit dans le manuel utilisateur de la version 1.7 d'`Odyssée` (voir [FP98]).

Ce rapport a pour but de présenter l'application de l'outil de différentiation automatique `Odyssée` à `THYC3D` (voir [BDRB93] pour une description détaillée) qui est un code opérationnel de themohydraulique développé à EDF-DER. Un premier travail de collaboration entre EDF et l'INRIA avait permis de valider à la fois la méthode (différentiation automatique) et l'outil `Odyssée` sur une maquette mono-dimentionnelle de `THYC3D` appelée `THYC1D`. D'autre part, il a montré l'intérêt incontestable du mode inverse par rapport au mode direct pour le calcul d'un gradient, ainsi que les difficultés engendrées par le stockage nécessaire à ce mode de dérivation. Les résultats de ce travail sont décrits dans un rapport [EGFRS96] ou plus succinctement dans [DEFG96].

De l'étude présentée dans ce rapport on peut déduire que l'outil `Odyssée` a été étendu avec succès au traitement en mode direct ou inverse d'un code opérationnel tel que `THYC3D`.
Pour le mode inverse, deux conclusions peuvent être tirées de cette étude. D'une part, les nouveaux algorithmes de linéaires cotangent combinés permettent de traiter un code opérationnel. C'est à dire que le problème de taille mémoire rencontré lors de la première étude a été résolu au prix d'une augmentation du temps d'exécution raisonnable. D'autre part, cette étude a montré la difficulté de valider un code linéaire cotangent (ou adjoint) de grande taille. Elle a permis aussi de dégager certaines règles de programmation d'un modèle écrit en vu de l'utilisation de la différentiation automatique en mode inverse.

Il faut noter qu'à notre connaissance, il n'existe aucune expérience de différentiation automatique de code opérationnel en mode inverse. En effet, les deux autres outils de différentiation automatique qui mettent en oeuvre le mode inverse (`adolc, tamc, padre2`) ne peuvent pas être appliqués sur de très grands codes. Une expérience de dérivation d'un code de grande taille a été réalisée à partir d'`Odyssée` dans le domaine de la météorologie (sur une partie du code `Meso-NH`). Ce travail est présenté dans [CG97].

Chapitre 2

THYC3D pour la Différentiation Automatique

Dans cette section, nous présentons les problèmes dûs à la structure du code initial rencontrés dans le code de THYC3D lors de sa dérivation. De tels problèmes peuvent se présenter dans de nombreux codes numériques et doivent donc être étudiés.

2.1 Utilisation implicite des pointeurs

Dans le code de THYC3D nous avons constaté des problèmes dûs à l'utilisation implicite de pointeurs dans le code.

L'analyse statique du morceau de code suivant (extrait de la routine misajour) par Odyssée:

```
DIMENSION QX (IJKM), QY (IJKM), QZ (IJKM)

ILONG = 3 * IJKM
CALL CUMUL ( QX , QX , 1. , ACCR(IDQX) , ILONG )
```

lui permet de déduire que `qx` est modifié. Or de par leur définition initiale modifier `qx` de longueur `3*ijkm` est équivalent à modifier `qx, qy, qz` chacun de taille `ijkm`. `Odyssée` ne peut automatiquement calculer cette information, nous avons donc modifié le code source pour que l'analyse faite par `Odyssée` soit correcte. Le code de `misajour` modifié est :

```
call CUMUL(QX,QX,1.,ACCR(IDQX),IJKM)
call CUMUL(QY,QY,1.,ACCR(IDQY),IJKM)
call CUMUL(QZ,QZ,1.,ACCR(IDQZ),IJKM)
```

Toutes les routines de `THYC3D` utilisant un tel stratagème ont été modifiées à la main.

Cette erreur a été difficile à trouver et le fait que `qy, qz` ne semblait pas être modifiés provoquait deux erreurs :

1. `Odyssée` ne sauvegardait pas leurs valeurs et la trajectoire n'était donc pas reproduite correctement,
2. `qy, qz` n'étaient pas actifs alors qu'ils auraient dû l'être.

Une analyse du code dite « analyse des pointeurs » permettrait de détecter ces problèmes syntaxiquement, mais elle est très coûteuse en temps et doit donc être utilisée à bon escient.

2.2 Aliasing par passage de paramètres

La dérivation en mode inverse d'une instruction est différents suivant que la variable modifiée est lue dans l'instruction ou non. `Odyssée` effectue les deux types de transformations explicitées sur une instruction simple dans les figures 2.1et 2.2.

Le code `THYC3D` utilise des routines de base qui effectuent des opérations vectorielles de la forme `y = x + ab`. Par exemple, la routine `cumul` a pour arguments `( XMP1 , XM , ROM , GM , N )` et calcule :
`xmp1(i) <= xm(i) + rom * gm(i), i=1,n`.

$$a^* = a^* + y^* b \tag{2.1}$$
$$b^* = b^* + y^* a \tag{2.2}$$
$$x^* = x^* + y^* \tag{2.3}$$
$$y^* = 0. \tag{2.4}$$
$$\tag{2.5}$$

FIG. 2.1: *La dérivée de l'instruction :* `y = x + ab`

$$a^* = a^* + y^* b \tag{2.6}$$
$$b^* = b^* + y^* a \tag{2.7}$$
$$y^* = y^* \tag{2.8}$$
$$\tag{2.9}$$

FIG. 2.2: *La dérivée de l'instruction :* `y = y + ab`

`Odyssée` génère donc pour chacune de ces routines une unique routine dérivée inverse de la forme 2.1. Par exemple, la routine `cumulcl` a pour arguments `(xmp1, xm, rom, gm, n, xmp1ccl, xmccl, romccl, gmccl)`
et calcule :

```
xmccl(i) = xmccl(i)+xmp1ccl(i), i=1,n
gmccl(i) = gmccl(i)+xmp1ccl(i)*rom, i=1,n
romccl = romccl+xmp1ccl(i)*gm(i), i=1,n
xmp1ccl(i) = 0., i=1,n
```

Or ces routines sont appelées dans le code avec le même argument effectif à la place de leurs arguments formels `x, y`. Par exemple, `cumul` est appelé de la manière suivante : `CALL CUMUL (BILA(IBQX), BILA(IBQX), DT, AUX(1,16), IJKM)`. et sa dérivée `cumulcl` est donc appelée comme suit :
`CALL cumulcl(bila(ibqx), bila(ibqx), dt, aux(1,16), ijkm,`
`: bilaccl(ibqx), bilaccl(ibqx), dtccl, auxccl(1,16)).`

Lors de l'évaluation de cet appel, `bilaccl(ibqx)` sera calculé par les lignes 3 et 4 du code présenté dans la figure 2.1. Sa valeur calculée en 3 sera donc écrasée en 4 par 0. et toute exécution donnera une dérivée nulle.

Pour l'instant pour remédier à cela, nous avons modifié à la main le code dérivé. Par exemple le code de `cumulcl` dérivé puis modifié est le suivant :

```
tmp = xmp1ccl(i)
xmp1ccl(i) = 0.
xmccl(i) = xmccl(i)+tmp
gmccl(i) = gmccl(i)+tmp*rom
romccl = romccl+tmp*gm(i)
```

Cette modification permet d'obtenir une dérivée de `cumul` exacte que ses premiers et second arguments soient le même pointeur ou non.

Une telle modification ne peut être généralisée à toutes les routines car elle introduirait un grand nombre de variables temporaires. Par contre introduire des variables intermédiaires dans le code original (dans les appels) semble être la bonne solution.

2.3 Les fonctions tabulées

Dans le code de `princp`, différentes familles de routines en boîte noire (dont le source n'est pas disponible) sont appelées; celles qui utilisent et modifient leurs arguments, et celles qui utilisent des globales allouées dynamiquement.

Pour qu'`Odyssée` réalise une analyse cohérente des entrées/sorties d'un programme, il doit calculer les entrées/sorties de chaque routine. Il peut créer automatiquement une entrée dans la base d'informations pour les routines en boîte noire en supposant qu'elles ne modifient que leurs arguments (pas les variables globales).

Parmi les fonctions de `Thétis`, celles qui mémorisent des vecteurs ou effectuent des calculs à partir de valeurs mémorisées ne peuvent être correctement analysées par `Odyssée`. `Odyssée` ne peut absolument pas "deviner" que `tvsetp` mémorise la valeur du vecteur pression qui sera ensuite utilisée. Si on ne fournit pas l'information à `Odyssée`, l'analyse qu'il fera des appels et donc du code est faussée. La définition

entrée/sortie des fonctions de mémorisation, de calcul de saturation ou de calcul dans les plans après mémorisation doit donc être déclarée.

De plus, ces fonctions mémorisent et accèdent à des globales qui ne sont pas dans des commons de THYC3D, mais dans des emplacements alloués dynamiquement. Il n'est donc pas possible de décrire leur comportement à partir des variables globales du système. Nous avons donc déclaré dans `princp` un common supplémentaire `memo` composé de cinq globales réelles de taille `ijkmp1` : `memo_p, memo_t, memo_tit, memo_h, memo_s` représentant les mémorisations de la pression, la température, le titre, l'enthalpie et l'entropie. Les fonctions de calcul à la saturation à partir d'un vecteur pression mémorisé `tvXXsm` peuvent alors être définies.

Par exemple, `tvsetpt` et `tvsigsm` sont définies par :

```
setdummys tvsetp (p1 p2 p3 p4 p5)
setglobals tvsetp (memo_p)
setinout tvsetp (p1 p2 p3) (p4 p5) () (memo_p)

setdummys tvsigsm (p1 p2 p3)
setglobales tvsigsm (memo_p)
setinout tvsigsm (p1) (p2 p3) (memo_p) ()
```

La définition des fonctions de calcul `tvXXvm` à partir de vecteurs mémorisés pose un problème supplémentaire. En effet, `tvttvm` calcule la température à partir de deux vecteurs mémorisés : la pression et l'un des trois autres vecteurs mémorisables. Le système ne peut déduire statiquement quel est l'autre vecteur qui a été mémorisé. Pour que l'analyse soit juste, nous avons défini par les fonctions `tvxxvm` en leur mettant en entrée toutes les globales de mémorisation.

```
setdummys tvsetpt (p1 p2 p3 p4 p5 p6 p7)
setglobals tvsetpt (memo_p memo_t memo_tit)
setinout tvsetpt (p1 p2 p3 p4 p5) (p5 p6 p7)
                 () (memo_p memo_t memo_tit)

setdummys tvttvm (p1 p2 p3)
setglobales tvttvm (memo_p memo_t memo_tit memo_h memo_s)
setinout tvttvm (p1) (p2 p3)
```

```
(memo_p memo_t memo_tit memo_h memo_s) ()
```

Chapitre 3

Dérivation manuelle des fonctions tabulées

L'utilisation dans THYC3D de fonctions tabulées (dont le source ne peut être lu) pose un problème lors de l'analyse du code par Odyssée. Ce problème est décrit dans la section 2.3, il a été résolu par la définition d'une base spécifique d'informations.

Le deuxième problème posé par l'utilisation de telles routines est le problème de la définition des routines dérivées associées. Dans ce chapitre, nous décrivons les problèmes posés ainsi que les solutions que nous avons apportées.

3.1 Principes généraux

Les routines C n'étant appelées qu'en cas de couplage avec COCCINELLE et le cas test à traiter ne prenant pas en compte ce couplage, leur dérivées n'ont pas été définies. Pour les dériver, il faudrait aller lire ou écrire dans un fichier les valeurs de dérivées transmises par ces canaux de communication spécifiques à UNIX appelés pipe.

Les dérivées des fonctions thermohydrauliques de Protée et de Thétis (TvXXsa, tvXXpt, tvXXps, tvXXph décrites succinctement dans la section 2.3) doivent être

définies. L'écriture des dérivées des fonctions de `Thétis` qui gèrent de la mémoire allouée dynamiquement a posé des problèmes. Nous appellerons `fonctions simples` les fonctions de `Protée` et de `Thétis` qui n'utilisent pas d'allocation dynamique et `fonctions memo` les autres. Les `fonctions memo` sont de deux sortes : les fonctions `tvsetpX` positionnent des variables `Thétis` à partir de variables `THYC3D`, et les fonctions `tvXXsm, tvXXvm` calculent des variables de `THYC3D` à partir de variables de `Thétis`.

Pour écrire les dérivées des fonctions `memo`, nous avons fait les hypothèses suivantes qui doivent être vérifiées pour que le code écrit soit correcte :

1. les routines `tvsetpX` ne font rien d'autre que mémoriser les valeurs qui leur sont passées,
2. les routines `tvsetpX` mémorisent la pression dans le même emplacement mémoire,
3. les routines `tvsetpX` n'écrasent pas les vecteurs mémorisés si on change de fluide,
4. les routines `tvXXvm` effectuent leurs calculs à partir des **deux derniers vecteurs** mémorisés.

Les dérivées des fonctions `memo` de calcul doivent calculer des différences divisées. C'est à dire qu'elles doivent calculer la fonction en deux points, entre ces deux appels, il faut donc avoir modifié les vecteurs mémorisés. Pour reproduire un calcul équivalent pour les dérivées à celui effectué pour les variables du code initial, il faut donc :

1. savoir quels vecteurs ont été positionnés dans le code initial car ces mêmes variables ainsi que leurs dérivées doivent être positionnées par la dérivée,
2. savoir pour quel numéro de fluide ces vecteurs ont été positionnés (pour le cas général),
3. pouvoir repositionner ces vecteurs à leur valeur initiale à la fin de la dérivée pour que les évaluations qui suivent soient correctes.

```
      subroutine  tvsetp_jum  (p1, p2, p3, p4, p5)
...
      COMMON /memo/memo_p, memo_t, memo_tit, memo_h, memo_s
      COMMON /memo_int/memo_nf, memo_trace
      integer memo_nf, memo_trace
      real memo_s(odyijkmp1), memo_h(odyijkmp1),
     : memo_tit(odyijkmp1),
     : memo_t(odyijkmp1), memo_p(odyijkmp1)

      memo_nf = p1
      memo_trace = 2
      do i=1, p2
         memo_p (i) = p3(i)
      end do
```

FIG. 3.1: *Routine jumelle de* `tvsetp`

Comme ces informations ne peuvent être extraites de `Thétis` à partir de `THYC3D`, nous avons dû les mémoriser dans des variables globales de `THYC3D`. Les variables globales de `THYC3D` nécessaires au calcul de ces dérivées sont donc :

`memo_p, memo_t, memo_tit, memo_h, memo_s` les copies des vecteurs mémorisés,

`memo_nf` le numéro de fluide pour lequel les vecteurs ont étés mémorisés,

`memo_trace` un indicateur des fonctions de mémorisation 0 pour `tvsetp`, 1 pour `tvsetps`, 2 pour `tvsetpt`, et 3 pour `tvsetph`.

Aux fonctions `memo` de positionnement, il faut donc ajouter une fonction jumelle qui positionne ces variables globales. La figure 3.1 présente le code à ajouter à `tvsetp`.

On peut noter que les variables allouées dynamiquement dans `Thétis` sont doublées lors de la dérivation par des copies (`memo_p`, ...). La gestion dynamique de `Thétis` n'a plus beaucoup d'intérêt dans ce cadre.

```
subroutine vmpstl (p1, p2, p3,
                   p1tl, p2tl, p3tl)
...

C calcul de la jacobienne
t=p1
if (t.eq.0.) t=1.
t=abs(sqrt(myeps)*t)
new_p3 = vmps (p1+t,p2)
jac_p3_p1 = (new_p3 - p3)/t

t=p2
if (t.eq.0.) t=1.
t=abs(sqrt(myeps)*t)
new_p3 = vmps (p1,p2+t)
jac_p3_p2 = (new_p3 - p3)/t

C calcul du lineaire tangent

p3tl = jac_p3_p1 * p1tl +
       jac_p3_p2 * p2tl

return
end
```

```
subroutine vmpscl (p1, p2, p3,
                     p1cl, p2cl, p3cl)
...

C calcul de la jacobienne
t=p1
if (t.eq.0.) t=1.
t=abs(sqrt(myeps)*t)
new_p3 = vmps (p1+t,p2)
jac_p3_p1 = (new_p3 - p3)/t

t=p2
if (t.eq.0.) t=1.
t=abs(sqrt(myeps)*t)
new_p3 = vmps (p1,p2+t)
jac_p3_p2 = (new_p3 - p3)/t

C calcul du lineaire cotangent

p1cl = p1cl + jac_p3_p1 * p3cl
p2cl = p2cl + jac_p3_p2 * p3cl
p3cl = 0.
return
end
```

FIG. 3.2: *Dérivée de la routine de* `Protée vmps`

3.2 Dérivées des fonctions simples

Nous avons expliqué comment écrire de telles dérivées dans le rapport précédant [EGFRS96]. Dans cette section, nous rappelons simplement les principes utilisés.

Les fonctions de `Protée` étant scalaires, nous avons généré leur linéaire tangent en écrivant le produit de la jacobienne calculé par différences divisées par le vecteur direction et le linéaire cotangent par le produit de la transposée de la jacobienne par le vecteur "direction" dans l'espace dual. Dans la figure 3.2 un exemple de telles dérivées est présenté.

Les fonctions de `Thétis` calculent des vecteurs de taille 279 dans notre cas test. Pour éviter un calcul de jacobienne toujours coûteux, leurs dérivées calculent directement

des différences divisées dans la direction choisie (voir formule (A.5) page 54 dans le rapport [EGFRS96]). En mode direct ce calcul se fait ligne par ligne, alors qu'en mode inverse il se fait colonne par colonne.

3.3 Dérivées des fonctions memo de positionnement

Les fonctions qui mémorisent les variables de THYC3D dans Thétis sont équivalentes à des affectations.

En mode direct, l'appel de la routine tvsetp qui positionne la pression devrait être dérivé en tvsetptl qui positionne la pression et sa dérivée par tvsetp. Mais on ne peut utiliser la même routine (tvsetp) pour positionner la pression et sa dérivée car celle-ci utilise un seul emplacement mémoire, ceci reviendrait donc à écraser une valeur par l'autre. Pour éviter cela, nous avons choisi de positionner la dérivée dans une variable globale de THYC3D, par exemple memo_pttl pour la pression.

De même, en mode inverse, l'appel de la routine tvsetp qui positionne la pression est dérivé en un calcul de l'adjoint de la pression à partir de memo_pccl l'adjoint de memo_p et qui remet l'adjoint de memo_p à zéro.

A titre d'exemple, la figure 3.3 présente les code des dérivées en modes direct et inverse de tvsetp.

3.4 Dérivées des fonctions memo de calcul

Les dérivées des fonctions memo de calcul utilisent comme les routines simples de Thétis des différences divisées directionnelles par ligne ou par colonne. La difficulté est qu'elles doivent positionner les deux points par les routines memo de positionnement. De plus, en fin de calcul, elles doivent repositionner le point initial. La figure 3.4 montrent le code des dérivées de tvsigsm. On peut noter que les fonctions norme_max, square_sum_sup, square_sum sont utilisées pour le calcul de t (dans 3.4). Ces fonctions sont décrites page 54 dans le rapport [EGFRS96] et calculent

```
subroutine tvsetptl (p1, p2, p3, p4, p5,
                     p3tl)
implicit none
include "odyparam.inc"
integer i
integer p1, p2, p4(odyijkmp1), p5
real p3(odyijkmp1), p3tl(odyijkmp1)

COMMON /memottl/memo_pttl, memo_tttl,
    memo_titttl, memo_httl, memo_sttl
integer memo_nf, memo_trace
real memo_sttl(odyijkmp1),
     memo_httl(odyijkmp1),
     memo_titttl(odyijkmp1),
     memo_tttl(odyijkmp1),
     memo_pttl(odyijkmp1)
integer tvsetp
external tvsetp

p5 = tvsetp (p1, p2, p3, p4)
do i=1, p2
   memo_pttl(i) = p3tl(i)
end do
return
end
```

```
subroutine  tvsetpcl  (p1, p2, p3, p4, p5,
                       p3cl)
implicit none
include "odyparam.inc"
integer i
integer p1, p2, p4(odyijkmp1), p5
real p3(odyijkmp1), p3tl(odyijkmp1)

COMMON /memoccl/memo_pccl, memo_tccl,
    memo_titccl, memo_hccl, memo_sccl
integer memo_nf, memo_trace
real memo_sccl(odyijkmp1),
     memo_hccl(odyijkmp1),
     memo_titccl(odyijkmp1),
     memo_tccl(odyijkmp1),
     memo_pccl(odyijkmp1)
integer tvsetp
external tvsetp

do i=1, p2
   p3cl(i) = p3cl(i) + memo_pccl(i)
   memo_pccl(i) = 0.
end do
return
end
```

FIG. 3.3: *Dérivée de la routine* `tvsetp` *en linéaire tangent et cotangent par rapport à* p3

respectivement :

$$norme\_max(l,x) = Max_{i=1,l} x_i \tag{3.1}$$

$$square\_sum(l,x) = \sum_{i=1,l} x_i^2 \tag{3.2}$$

$$square\_sum\_sup(l,xtl,x,max) = \sum_{\substack{i=1,l \\ |xtl(i)| \geq max}} x_i^2 \tag{3.3}$$

```
subroutine tvsigsmtl (p1, p2, p3,
                      p2tl)
...

p3 = tvsigsm (p1, p2)

C --- Calcul de t
max_tl = norme_max (p1, memo_pttl)
if (max_tl.eq.0.) then
   do i=1, p1
      p2tl(i) = 0.
   end do
   return
end if
max_tl = max_tl*myeps
t = square_sum_sup (p1, memo_pttl,
                    memo_p, max_tl)
if (t.eq.0.) t=1.
t = abs (sqrt(myeps*t/
          (square_sum(p1,memo_pttl))))

C --- Calcul des deux points
do i=1, p1
   tt = t*memo_pttl(i)
   pplus(i) = memo_p(i) + tt
   pmoins(i) = memo_p(i) - tt
end do

C --- Evaluation de la fonction
aux4=tvsetp (memo_nf,p1,pplus,aux3)
p3plus = tvsigsm (p1, p2plus)

C --- Evaluation de la fonction
aux4=tvsetp (memo_nf,p1,pmoins,aux3)
p3moins = tvsigsm (p1, p2moins)

C --- Calcul de la derivee
do i=1, p1
   p2tl(i) =
     (p2plus(i) - p2moins(i)) / (2.*t)
end do

C --- Restauration des donnees

aux4 = tvsetp (memo_nf,p1,memo_p,aux3)
return
end
```

```
subroutine tvsigsmcl (p1, p2, p3,
                      p2cl)
...

do i=1, p1

C --- Calcul de t
   t = memo_p(i)
   if (t.eq.0.) t=1.
   t=abs(sqrt(myeps)*t)/2



C --- Calcul des deux points
   do j=1, p1
      pmoins (j) = memo_p(j)
      pplus (j) = memo_p(j)
   end do
   pmoins (i) = memo_p(i) - t
   pplus (i) = memo_p(i) + t



C --- Evaluation de la fonction
   aux4=tvsetp(memo_nf,p1,pplus,aux3)
   p3plus = tvsigsm (p1, p2plus)

C --- Evaluation de la fonction
   aux4=tvsetp(memo_nf,p1,pmoins,aux3)
   p3moins = tvsigsm (p1, p2moins)

C --- Calcul de la dérivée
   do j=1, p1
      memo_pccl(i) = memo_pccl(i) +
           (p2plus(i) - p2moins(i)) /
           2*t* p2cl(i)
   end do
end do
do j=1, p1
   p2cl(j) = 0.
end do
return
end
```


FIG. 3.4: *Code des dérivées en mode direct et inverse de* `tvsigsm`

Chapitre 4

Dérivation de THYC3D par Odyssée

Le problème posé par EDF était de calculer :

- en mode direct, la sensibilité de toutes les composantes du titre vapeur `tit` par rapport à la composante en z de la vitesse relative entre phases `parcor(11,1,1)`,
- en mode inverse, la sensibilité du titre vapeur `tit(2,2,30)` par rapport à toutes les composantes de la vitesse relative entre phases `parcor`.

`Odyssée` dérive une branche d'un programme, il nous a donc fallu choisir dans le programme `thyc` à partir de quel niveau appliquer la dérivation. Nous avons choisi de dériver `thyc` (voir son graphe d'appel figure 4.1) à partir de `princp` car, avant cela, le programme fait l'allocation dynamique de certains tableaux ainsi que des initialisations.

Pour dériver le code `THYC3D` que ce soit en mode direct ou inverse, il nous a fallu suivre la méthodologie suivante :

1. Dériver le programme dont la routine de tête est `princp` par rapport à `parcor`.
2. Modifier le programme dérivé automatiquement.
3. Modifier le programme `thyc` pour allouer la mémoire nécessaire aux dérivées et appeler la routine `princpcl` au lieu de `princp`.

```
thyc +- alendy +- (i4tady)
     +- alplaq
     +- alredy +- (retady)
     +- arret
     +- debmes +- (dtime)
               +- reacar
               +- (time)
     +- debtem +- (ctime)
               +- (dtime)
               +- reacar
               +- (secnds)
               +- (time)
     +- dimcor +- arret
               +- rechmo +- arret
     +- dimsin +- rechmo +- arret
     +- info +- infosp
             +- lejour +- (ctime)
                       +- (date)
                       +- (time)
     +- inizer
     +- inpoin +- ecrdon +- arret
                         +- wrdon1
                         +- wrdon2
                         +- wrdon3
                         +- wrdon4
                         +- wrdon5
                         +- wrdon6
               +- leccrm +- arret
                         +- rechmo +- arret
               +- lecdon +- arret
                         +- lectur +- dekode +- carlu +- ipreca
                                             |        +- next
                                             +- cmd +- ipreva
                                             |      +- keywrd
                                             |      +- next
                                             +- dico +- keywrd
                                             +- intlu +- ipreva
                                             |        +- next
                                             +- iprev
                                             +- ipreva
                                             +- lgclu +- next
                                             +- next
                                             +- realu +- ipreva
                                                      +- next
               +- lecgec +- arret
                         +- rechmo +- arret
     +- lecges +- rechmo +- arret
     +- ouvrir
     +- princp +- ....
```


FIG. 4.1: *Graphe d'appel de* THYC3D

4. Écrire à la main les dérivées des fonctions tabulées.

Dans le chapitre précédent, nous avons décrit comment les dérivées des fonctions tabulées devaient être écrites. Dans ce chapitre, nous allons donc développer les autres étapes.

4.1 Dérivation de `princp` par rapport à `parcor`

Pour pouvoir dériver `princp`, différents fichiers sont nécessaires :

1. Deux fichiers de déclaration de la base d'informations : `thetis.bi` et `protee.bi`.
2. Un fichier de chargement : `thyc3d_load.batch`

```
load (THYC/coeur/calc/bifptu \
THYC/coeur/calc/bilcax\
THYC/coeur/calc/bilcpl\
...
)
```

3. Un fichier script : `thyc3d.batch`

```
setvar do_indentation 0
setvar if_indentation 0

setvar fortran_dir .
setvar ibasis_dir .
setvar include_path (THYC/include)

execbatch thyc3d_load
loadibasis thetis
loadibasis protee

diff -tl -h princp -vars parcor -split -nolib
diff -cl -goto -opt solv -h princp -vars parcor -split -nolib
quit
```

La commande `diff -tl` génère le linéaire tangent, et `diff -cl -goto -opt solv` génère le linéaire cotangent en utilisant l'algorithme qui traite les `goto` et en optimisant la boucle sur `solv`.

La dérivation de `THYC3D` consiste alors en l'exécution du fichier script par `Odyssée`:

```
          ODYSSEE
  copyright INRIA - UNSA
 Version 1.6 - Fri Oct 25 11:28:42 MET DST 1996

Sourcing .odysseerc...
OK
ODYTOP> execbatch thyc3d
```

La dérivation de `princp` prend environ 4h de temps `cpu` et la taille du code généré est de 80607 lignes, alors que le code initial fait 61725 lignes. Lors de la dérivation de `princp` sur un `dec alphastation 500` dont les caractéristiques sont `256 Mo - 333 Mhz - disk 2 gigas`, l'exécutable d'`Odyssée` grossit jusqu'à atteindre `154M`. Si la dérivation est faite en mode direct et inverse lors de la même session, le temps total est diminué d'autant puisque l'analyse du code n'est faite qu'une seule fois.

4.2 Modification manuelle du code dérivé

`Odyssée` crée des `parameter`s qu'il utilise comme taille de tableaux. Ces `parameter`s sont de la forme `odyijkmp1` et doivent avoir pour valeur la valeur de `ijkmp1`. Le système ne pouvant automatiquement les affecter à leur valeur correcte, ils sont affectés à la valeur 10 par défaut.

Il faut donc modifier le code généré pour affecter ces variables à leur valeur effective. Nous avons remplacé ces déclarations par l'inclusion du fichier `odyparam.inc` (voir figure 4.2). On peut remarquer que certains `parameter`s tels `odyijknz` devraient être affectés à 0, mais le compilateur `Fortran 77` n'accepte pas les déclarations de tailles de tableaux de la forme `(1:0)`, ces variables ont donc été affectées à 1.

```
INTEGER odyim
INTEGER odyjm
INTEGER odykm
INTEGER odyijkm
PARAMETER (odyim = 2)
PARAMETER (odyjm = 2)
PARAMETER (odykm = 30)
PARAMETER (odyijkm = odyim*odyjm*odykm)

INTEGER odyimp1
INTEGER odyjmp1
INTEGER odykmp1
INTEGER odyijkmp1
PARAMETER (odyimp1 = odyim +1)
PARAMETER (odyjmp1 = odyjm +1)
PARAMETER (odykmp1 = odykm +1)
PARAMETER (odyijkmp1 = odyimp1 * odyjmp1 * odykmp1)

INTEGER odylvarsp
PARAMETER (odylvarsp =   2)
INTEGER odyijknz
PARAMETER (odyijknz =   1)
```

FIG. 4.2: *Fichier de définition des* parameter*s créés par* Odyssée

De plus, `Odyssée` ne sait pas dériver les instructions `read`. Nous avons donc modifié le code de `leccltl` dans lequel la lecture de `tabval` écrase une variable active, pour écraser la dérivée de cette variable en l'initialisant à zéro.

```
READ (nfic, *, end = 44, err = 44), tabval
do myi=1, NBVAL
   tabvalttl(myi) = 0.
end do
```

4.3 Écriture de la routine principale

Pour écrire les codes de `thyctl` et `thyccl` qui calculent respectivement le linéaire tangent et le linéaire cotangent de `THYC3D` par rapport à `parcor`, nous avons modifié le programme principal initial `thyc`. Les modifications montrées par la suite sont semblables en mode direct et inverse, nous allons donc les expliquer pour le mode direct.

Le programme `thyc` calcule la taille mémoire nécessaire aux variables entières `ia` et aux variables réelles `a` et l'alloue dynamiquement par l'intermédiaire du morceau de code suivant :

```
      M = 0
      CALL ALENDY(M,IA,ITOTI)
c
      N = 0
      CALL ALREDY(N,A,ITOTR).
```

Nous avons ajouté à cette allocation celle de la mémoire nécessaire aux variables réelles dérivées que nous avons maximisée en prenant la taille mémoire réelle initiale :

```
      myn = 0
      call ALREDY(MYN,ATTL,ITOTR)
```

ou

```
myn = 0
call ALREDY(MYN,ACCL,ITOTR)
```

Enfin, nous avons remplacé l'appel de `princp` par celui de sa dérivée `princptl` (en mode direct) ou `princpcl` (en mod einverse) en ajoutant aux arguments initiaux les arguments dérivés nécessaires.

En initialisant les variables dérivées (direction en mode direct et variables duales en mode inverse) et en faisant imprimer à la sortie de `princptl` le vecteur dérivé souhaité, toutes les sensibilités peuvent être obtenues.

Chapitre 5

Validation des résultats

Nous avons testé les deux dérivées `thyctl` et `thyccl` décrites précédemment sur plusieurs jeux de données. Dans ce document, nous présentons les résultats obtenus pour un jeux de donné dont les données principales de pression en sortie du coeur modélisé en 1D et d'enthalpie entrée(-sortie, la valeur en sortie n'étant prise en compte qu'en cas de recirculation avale) sont les suivantes : $P = 7000000.$ et $1267000. < H (< 1270000.)$. La simulation dure 600 secondes, nous avons choisi un pas de temps 0.05s ce qui produit 1200 itérations.

Pour ce jeu de données, nous avons calculé :

- en mode direct, la sensibilité de toutes les composantes du titre vapeur `tit` par rapport à la composante en z de la vitesse relative entre phases `parcor(11,1,1)`,
- en mode inverse, la sensibilité du titre vapeur `tit(2,2,30)` par rapport à toutes les composantes de la vitesse relative entre phases `parcor`.

Pour le jeux de données présentes ici, le nombre de composantes du titre vapeur `tit` est 279, alors que le nombre de composantes de la vitesse relative est 1000.

i	j	k	$\partial tit(i,j,k)/\partial parcor(11,1,1)$
2	2	2	−1.1260826554220D-04
2	2	3	−7.6277673010123D-03
2	2	4	−1.5020730080338D-02
2	2	5	−2.2292784570842D-02
2	2	6	−2.9444780335587D-02
2	2	7	−3.6477301845734D-02
2	2	8	−4.3390765419916D-02
2	2	9	−5.0185476373978D-02
2	2	10	−5.6861664272322D-02
2	2	11	−6.3419505472144D-02
2	2	12	−6.9859138022208D-02
2	2	13	−7.6180671802554D-02
2	2	14	−8.2384195574458D-02
2	2	15	−8.8469781993612D-02
2	2	16	−9.4437491248784D-02
2	2	17	−1.0028737375017D-01
2	2	18	−0.10601947213200
2	2	19	−0.11163382278399
2	2	20	−0.11713045702968
2	2	21	−0.12250940203052
2	2	22	−0.12777068151938
2	2	23	−0.13291431636853
2	2	24	−0.13794032503930
2	2	25	−0.14284872396093
2	2	26	−0.14763952783701
2	2	27	−0.15231274988610
2	2	28	−0.15686840205062
2	2	29	−0.16130649516705
2	2	30	−0.16562666957653

FIG. 5.1: *Dérivées partielles non nulles calculées par le linéaire tangent.*

5.1 Résultats des tests en valeur

Les dérivées partielles calculées par le code linéaire tangent de THYC3D sont les suivantes :

$$\forall (i,j,k) \in [1..3] \times [1..3] \times [2..30] \ \frac{\partial tit(i,j,k)}{\partial parcor(11,1,1)}$$

et celles calculées par le linéaire cotangent sont :

$$\forall (i,j,k) \in [1..50] \times \times [1..20] \ \frac{\partial tit(2,2,30)}{\partial parcor(i,j,k)}$$

i	j	k	$\partial tit(2,2,30)/\partial parcor(i,j,k)$
1	1	4	−2.1449377318471D-14
2	1	1	−9.4131465415044D-19
5	1	4	1.7943094373335D-06
11	1	1	−0.16562666957722
14	1	4	−5.2629901134261D-03

FIG. 5.2: *Dérivées partielles non nulles calculées par le linéaire cotangent.*

Le tableau 5.1 (page 30) présente la valeur des dérivées partielles non nulles obtenues par le linéaire tangent et le tableau 5.2 (page 31) montre celles obtenues par le linéaire cotangent.

Pour vérifier ces valeurs, nous avons calculé la dérivée partielle commune

$$\partial tit(2,2,30)/\partial parcor(11,1,1)$$

par différences divisées pour un pas "optimal" (voir la figure 5.3 (page 32) pour le choix de ce pas optimal).

pas	$\partial tit(2,2,30)/\partial parcor(11,1,1)$
1.e-7	-0.16563039995088
1.e-6	-0.16563032997907
1.e-5	-0.16562916599905
1.e-4	-0.16561744229981
1.e-3	-0.16550029545997
1.e-2	-0.16433788056900
1.e-1	-0.15355305792100

FIG. 5.3: *Choix d'un pas optimal pour les différences finies décentrées.*

différences finies décentrées	linéaire tangent	linéaire cotangent
-0.16562916599905	-0.16562666957653	-0.16562666957722

FIG. 5.4: *Valeurs de* $\partial tit(2,2,30)/\partial parcor(11,1,1)$.

La figure 5.4 (page 32) présente les valeurs de la dérivée partielle

$$\partial tit(2,2,30)/\partial parcor(11,1,1)$$

calculée à la fois en linéaire tangent et cotangent comparée à celle obtenue par différence divisées. On constate que les valeurs de cette dérivée partielle calculées en linéaire tangent et linéaire cotangent ont 12 chiffres significatifs communs et qu'elles semblent correctes vis à vis de celle obtenue par différences finies.

Les tests effectués montrent une concordance (plus de 12 chiffres) entre les valeurs obtenues par différentiation automatique et par différences finies. Nous pouvons donc

en déduire d'une part que le code généré par Odyssée est correct, d'autre part que le code écrit à la main pour les fonctions en boîte noire est correct.

5.2 Résultats des tests en temps et espace

Dans cette section, nous comparons le code de THYC3D et ses codes dérivés. Le temps d'exécution est labellé CPU en secondes, et la taille des exécutables est labellée runtime, externe en Méga octets. On peut noter que runtime est la taille maximum de l'exécutable lors de l'exécution, et externe la taille des fichiers écrits sur disque lors de l'exécution.

Le tableau suivant présente une comparaison des résultats en espace mémoire et temps de calcul de THYC3D, de son linéaire tangent et de son linéaire cotangent. La première ligne correspond aux valeurs en temps et espace, la deuxième correspond aux ratios de ces valeurs par rapport à la fonction initiale.

fonction initiale		linéaire tangent		linéaire cotangent		
CPU	size	CPU	size	CPU	size	externe
16.12	23.5	39.72	33	847.69	60	960
		2.46	1.4	52.58	2.5	

Le linéaire tangent de THYC3D calcule une colonne de la jacobienne et donc 279 dérivées partielles. On peut constater d'après le tableau précédant que le temps d'exécution du linéaire tangent est deux fois et demi (2.46) celui de la fonction initiale. De même, la taille du linéaire tangent est deux fois (1.4) supérieure à celle de la fonction.

Le linéaire cotangent de THYC3D calcule une ligne de la jacobienne et donc 1000 dérivées partielles. La taille totale du linéaire cotangent de THYC3D est de 1020 Méga octets si on sauve les variables calculées à tous (1200) les pas de temps. Ce code n'est donc pas exécutable sans la méthode de « checkpoint » qui permet de stocker seulement certaines parties de la trajectoire. L'utilisation de checkpoints a permis de contrôler la taille de l'exécutable (en faisant certaines sauvegardes sur le disque) pour permettre son utilisation sur une machine de taille normale. Le ratio en taille de l'exécutable (runtime) de la dérivée par rapport à la fonction est de 2.5 ce qui est l'objectif que nous nous étions fixé (entre 2 et 3).

Lors de cette étude, nous n'avons pas cherché à optimiser le code cotangent en temps ce qui explique que le temps d'exécution du linéaire tangent est 52.58 fois celui de la fonction. Ceci est beaucoup plus que le ratio théorique de 5 atteignable pour les programmes sans boucle et valide pour la plupart des applications que nous avons étudiées jusqu'à présent ([DEFG96], [CG97], [MRSM96], [FD98]).

Dans la suite de cette section, nous cherchons à expliquer ce ratio important en décomposant le temps de calcul de l'inverse. Nous pensons à priori que ces ratios en temps d'exécution et en espace mémoire du linéaire cotangent peuvent être diminués en approfondissant l'étude dans quatre directions :

1. L'algorithme général de mode inverse est tel que dans le code généré, chaque routine originale est exécutée un nombre de fois égal à sa profondeur dans le graphe d'appel (la profondeur maximale est de 10 pour `THYC3D`). En instrumentant le code, nous avons calculé que le temps de calcul cumulé des parties progressives était de 430 secondes en incluant les sauvegardes (allouées dynamiquement). Ceci correspond à la moitié du temps de calcul du code cotangent.

2. L'algorithme utilisé par la dérivée inverse dynamiquement le graphe de flot. Ceci a deux conséquences : d'une part le compilateur ne peut pas vraiment optimiser le code, d'autre part l'allocation dynamique d'espace mémoire peut être coûteuse en temps, en particulier à l'écriture. Le compilateur ne peut probablement pas optimiser le code car à chaque instruction originale (ou dérivée) est associée un appel externe. D'autre part, en instrumentant le code, nous avons calculé que les temps cumulés d'écriture et de lecture des sauvegardes (dans l'algorithme dynamique) sont de 168s pour l'écriture et 177s pour la lecture.

3. Les dérivées des fonctions tabulées calculent la jacobienne complète puis font le produit par la direction dans l'espace dual. En instrumentant le code, nous avons calculé que la dérivée d'une fonction tabulée prend (en moyenne) 40 fois plus de temps de calcul que la fonction initiale. Et que le temps cumulé des 33600 appels d'une fonction tabulée et de sa dérivée est de 177 secondes. Ceci correspond à un quart du temps de calcul total du code cotangent.

4. Les dérivées des solveurs linéaires générées automatiquement mémorisent les valeurs calculées avant convergence, ce qui peut nécessiter beaucoup de mémoire (et donc du temps d'accès à cette mémoire).

La table suivante récapitule la décomposition du temps de calcul du linéaire tangent :

	Temps (s)	Temps/848(%)
Recalcul de la fonction	262	31 %
Écriture dynamique des sauvegardes	168	20 %
Lecture dynamique des sauvegardes	177	21 %
Dérivées par Différences Divisées	177	21 %
Dérivées par Différentiation Automatique	64	8 %
Calcul du linéaire cotangent	848	100 %

En regardant les différents temps obtenus on peut voir que le temps de calcul des parties régressives du code cotangent (sans le temps d'exécution des dérivées des fonctions tabulées) est de :

$$848 - 430 - 177 - 177 = 64 \text{ secondes.}$$

Le ratio en temps d'exécution des parties adjointes par rapport à la fonction est $64/16 = 4$ ce qui correspond au ratio théorique de 5. L'importance du ratio en temps du code généré par rapport au code initial ne vient donc pas du calcul des variables dérivées (qui ne prend que 8% du temps total), mais plutôt des recalculs de la fonctions et de la sauvegarde dynamique des valeurs calculées par le code initial. De plus, on peut voir que la dérivation de fonctions tabulées à en mode inverse un coût important qui n'apparaît pas en mode direct.

5.3 Conclusion

Il est à noter qu'aucune expérience de Différentiation Automatique en mode inverse d'un code opérationnel complet utilisant un autre outils de différentiation automatique qúOdyssée n'a été publiée jusqu'à présent. Une expérience de dérivation d'un code de grande taille a été réalisée à partir d'Odyssée dans le domaine de la météorologie. Ce travail de génération "automatique" de l'adjoint d'une partie de Meso-NH (code météo développé au CNRM) est présenté dans [CG97]. L'obtention de l'adjoint correcte d'un tel code en trois mois (y compris les optimisations à la main) est un exploit. Les optimisations du code généré par Odyssée effectuées manuellement ont eu pour but principal de limiter les sauvegardes. D'une part, les sauvegardes des parties linéaires ont été supprimées, d'autre part les variables d'état ont été identifiées et des recalculs des autres variables modifiées ont été introduits.

Par cette étude, nous avons prouvé qu'un code de taille industrielle tel que `THYC3D` pouvait être dérivé entièrement automatiquement (par `Odyssée`). De plus, même en mode inverse le code généré est exécutable sur une station de travail. Nous pouvons donc en conclure que de la différentiation automatique est applicable à un code industriel quelconque puisque `THYC3D` n'a pas été écrit en prévoyant qu'il serait dérivé automatiquement.

Maintenant que la faisabilité de la différentiation automatique a été prouvée sur un code industriel quelconque, nous allons étudier tous les moyens d'optimiser le code généré en conformité avec les observations faites dans la section précédente. Nous allons en particulier travailler sur les algorithmes de dérivation et de sauvegarde des valeurs calculées par la fonction.

Le second résultat de cette étude est de pouvoir mieux décrire des normes de programmation d'un modèle pour faciliter l'utilisation de la Différentiation Automatique. Une ébauche de ces règles de programmation peut être déduite de la section 2 qui décrit les problèmes rencontrés sur `THYC3D`. Une première règle d'or est : un code « bon pour la dérivation automatique » ne doit en aucune manière masquer les dépendances entre les variables.

Bibliographie

[BDRB93] E. Briere, F. David, P. Rascle, et P-J. Bonamy. « THYC V3.0 : Modélisation et Méthodes Numériques ». Rapport HT-13/92056B, HT-33/92.11B, EDF/DER, février 1993.

[CG97] I. Charpentier et M. Ghemires. « Génération automatique de codes adjoints : Stratégies d'utilisation pour le logiciel Odyssée, Application au code météorologique Meso-NH ». Rapport de recherche 3251, INRIA, septembre 1997.

[DEFG96] C. Duval, P. Erhard, C. Faure, et JC. Gilbert. « Application of the Automatic Differentiation Tool Odyssée to a system of thermohydraulic equations. ». Dans J.-A. Désidéri, P. Le Tallec, E. Oñate, J. Périaux, et E. Stein, éditeurs, *Proc. of ECCOMAS'96*, volume Numerical Methods in Engineering'96, pages 795–802. John Wiley & Sons, septembre 1996.

[EGFRS96] F. Eyssette, JC. Gilbert, C. Faure, et N. Rostaing-Schmidt. « Applicabilité de la différentiation automatique à un système d'équations aux dérivées partielles régissant les phénomènes thermohydrauliques dans un tube chauffant. ». Rapport de recherche 2795, INRIA, février 1996. EDF/DER , HT-13/96/001/A.

[EV96] F. Eyssette et F. Vyskocil. « Roundoff Errors Propagation Analysis with Automatic Differentiation of Fortran 77 Code ». Dans J.-A. Désidéri, P. Le Tallec, E. Oñate, J. Périaux, et E. Stein, éditeurs, *Numerical Methods in Engineering'96*, pages 1018–1021. John Wiley & Sons, septembre 1996.

[Fau96] C. FAURE. « Splitting of Algebraic Expressions for Automatic Differentiation. ». Dans Martin BERZ, Christian BISCHOF, George CORLISS, et Andreas GRIEWANK, éditeurs, *Computational Differentiation : Techniques, Applications, and Tools*, pages 117–127. SIAM, Philadelphia, Penn., 1996.

[FD98] C. FAURE et C. DUVAL. « Automatic Differentiation for Sensitivity Analysis. A test case. ». Dans K. CHAN, S. TARANTOLA, et F. CAMPOLONGO, éditeurs, *Proceedings of Second International Symposium on Sensitivity Analysis of Model Output (SAMO'98)*, volume EUR report 17758. EN, Luxembourg, 1998.

[FP97] C. FAURE et Y. PAPEGAY. « Odyssée Version 1.6. The language reference manual. ». Rapport technique 3251, INRIA, novembre 1997.

[FP98] C. FAURE et Y. PAPEGAY. « Odyssée User's Guide. Version 1.7 ». Rapport technique 0224, INRIA, septembre 1998.

[Gil92] J.C. GILBERT. « Automatic Differentiation and iterative processes ». *Optimization Methods and Software*, 1:13–21, 1992.

[GVM91] J.C. GILBERT, G. Le VEY, et J. MASSE. « La Différentiation Automatique de fonctions représentées par des programmes ». Rapport de recherche 1557, INRIA, novembre 1991.

[Has96] E. HASSOLD. « Automatic Differentiation applied to a Nonsmooth Optimization Problem ». Dans J.-A. DÉSIDÉRI, P. LE TALLEC, E. OÑATE, J. PÉRIAUX, et E. STEIN, éditeurs, *Numerical Methods in Engineering'96*, pages 835–841. John Wiley & Sons, septembre 1996.

[MRSM96] J.-M. MALÉ, N. ROSTAING-SCHMIDT, et N. MARCO. « Automatic Differentiation: an Application to Optimum Shape Design in Aeronautics ». Dans J.-A. DÉSIDÉRI, C. HIRSCH, P. LE TALLEC, E. OÑATE, M PANDOLFI, J PÉRIAUX, et E. STEIN, éditeurs, *Minisymposia of ECCOMAS 96*. John Wiley & Sons, Ltd, septembre 1996.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399